Project 5

Ann Keenan (akeenan2)

Results

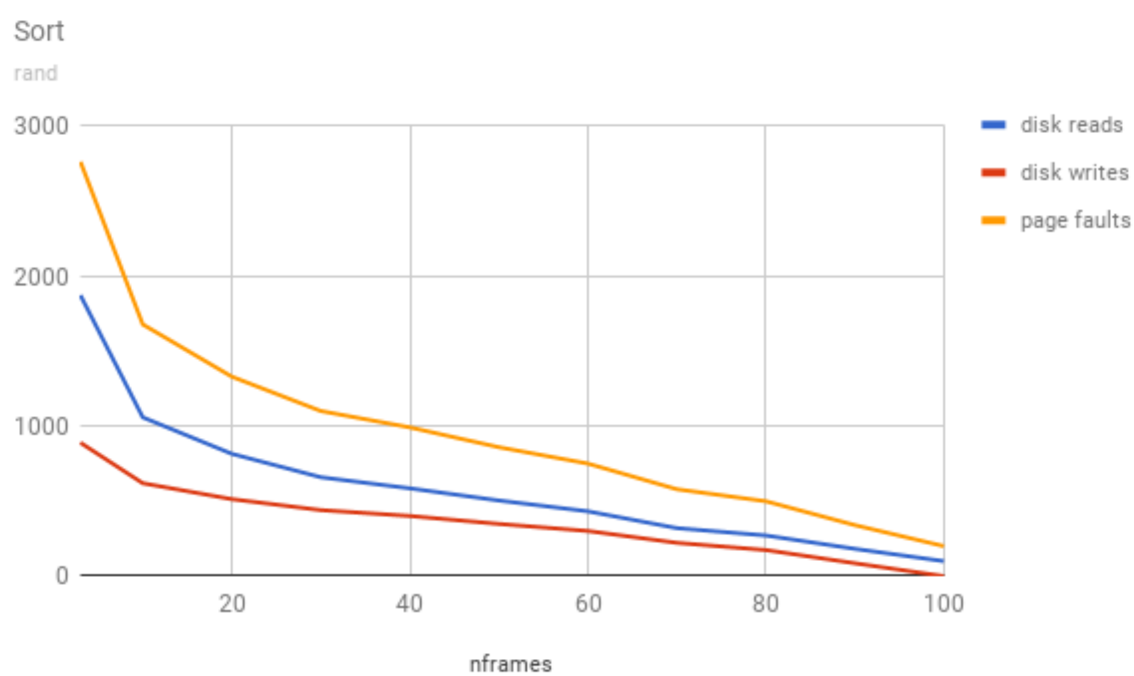
The following tests were run with 100 pages and 75, 50 or 25 frames. The program was run on student00, which has twelve of the Six-Core AMD Opteron(tm) Processor 2431, with 6 cpu cores each and a cpu MHz of 2393.717. The command line arguments run were of the format: `./virtmem 100 75 rand sort`.

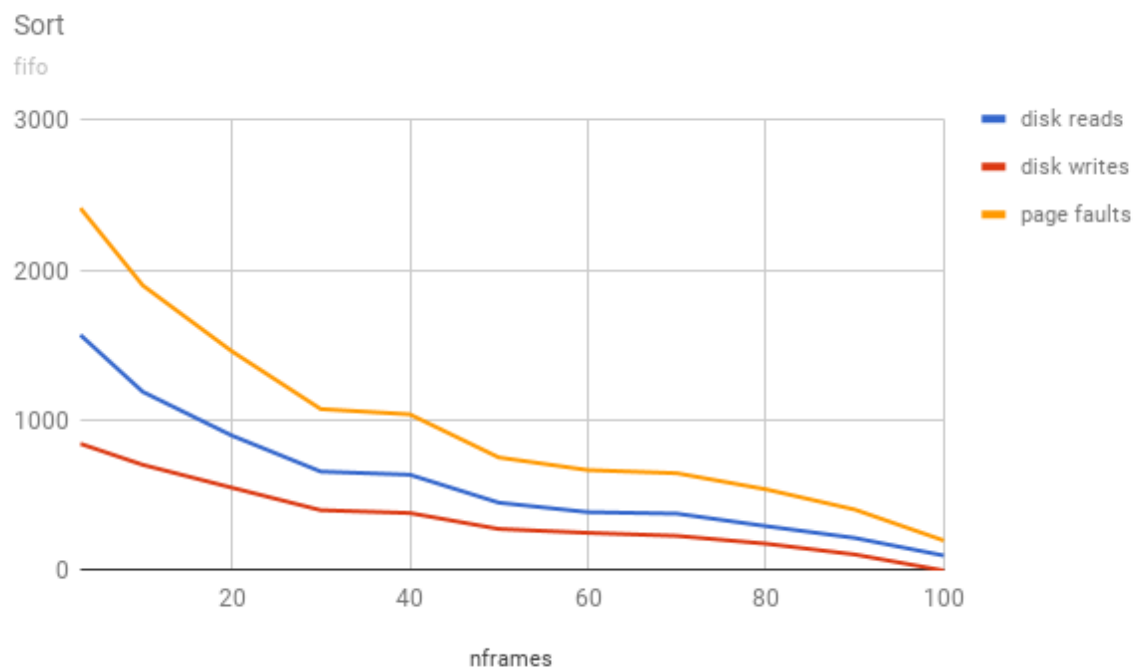
A shell script was written to run multiple commands, `run_virtmem.sh`, which can be run with the command: `./run_virtmem.sh rand sort` in order to run the virtmem function with the rand algorithm and sort program with 100 pages and between 3-100 frames.

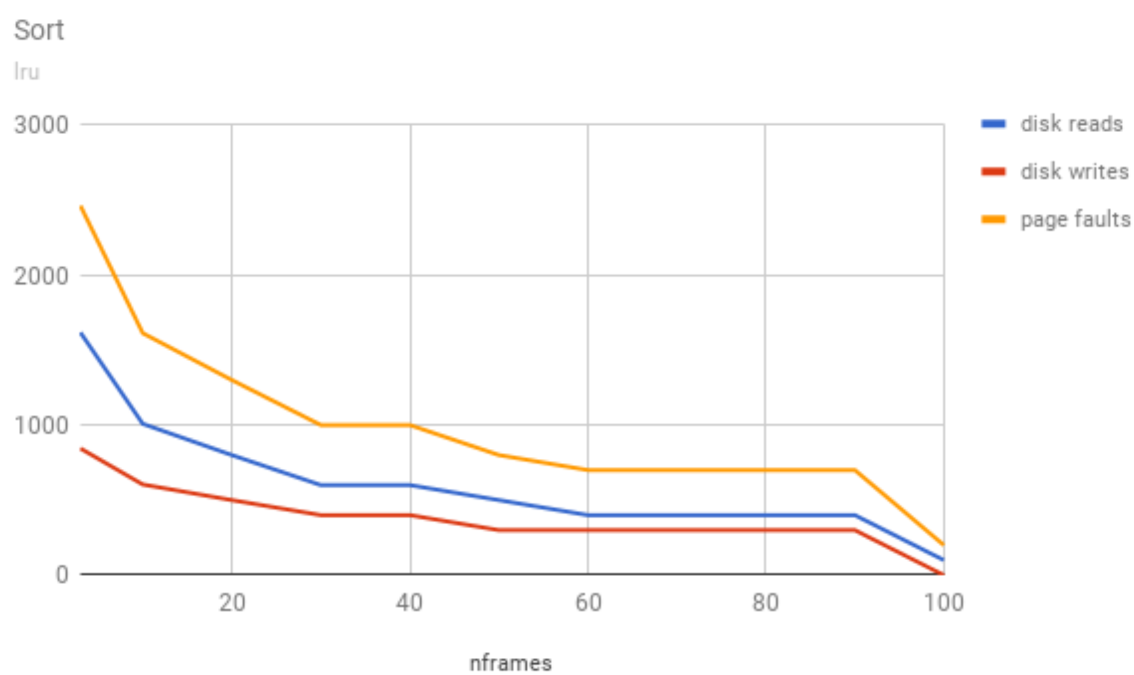
Further arguments run changed out `rand` for `fifo` or `custom` and `sort` for `scan` or `focus`.

Graphs

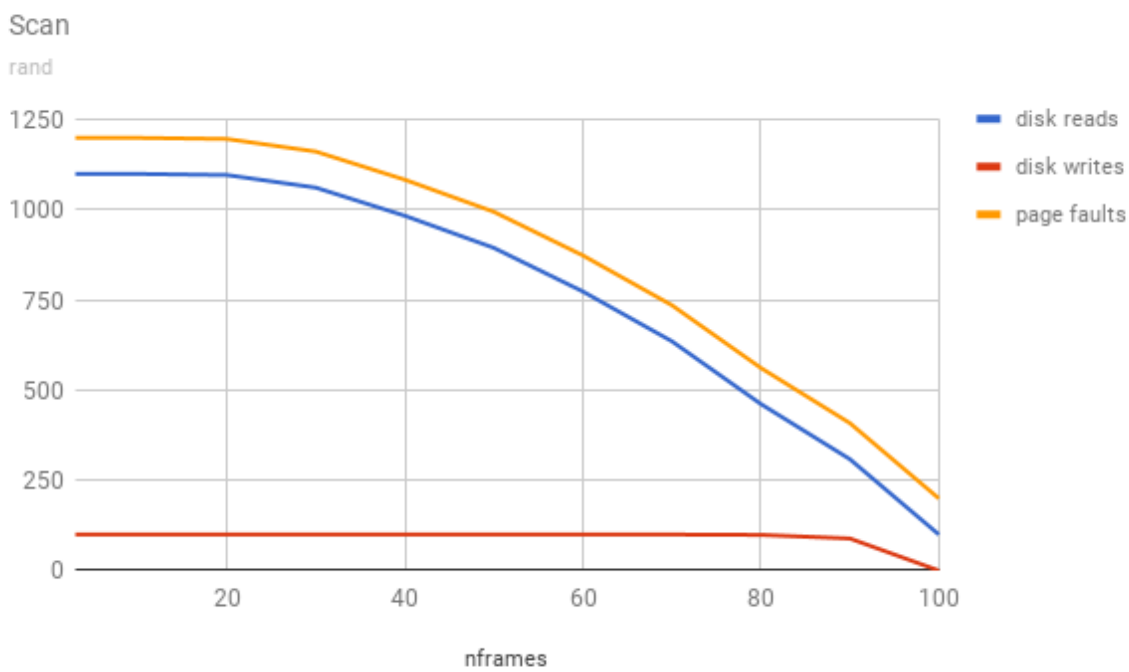
Sort

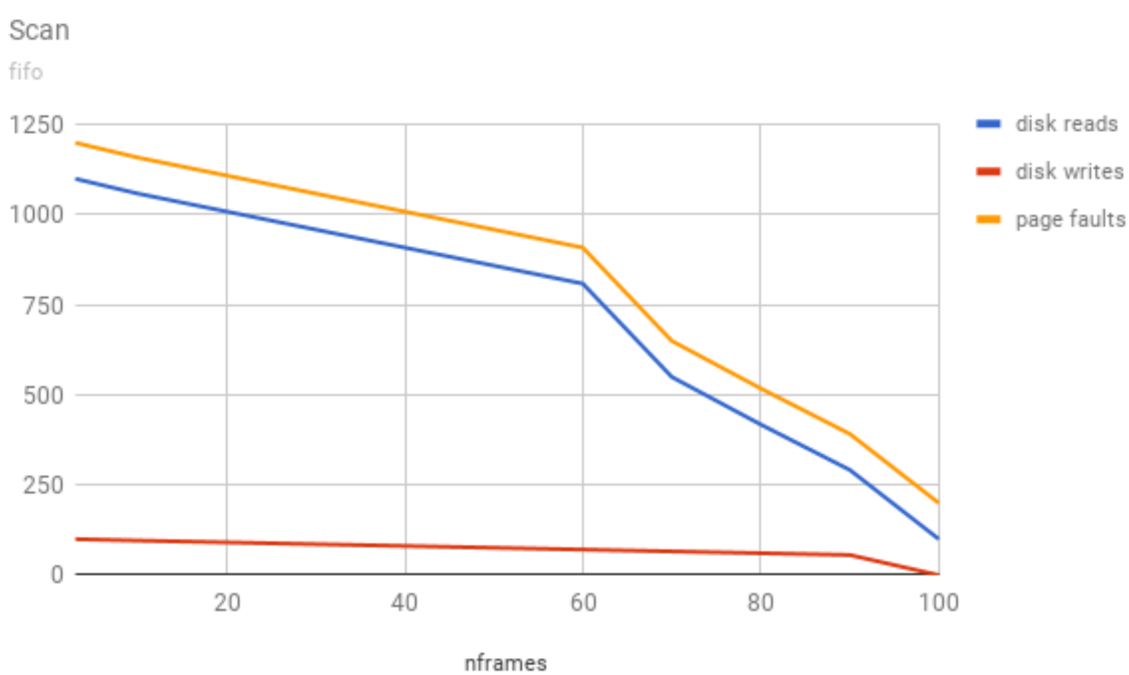


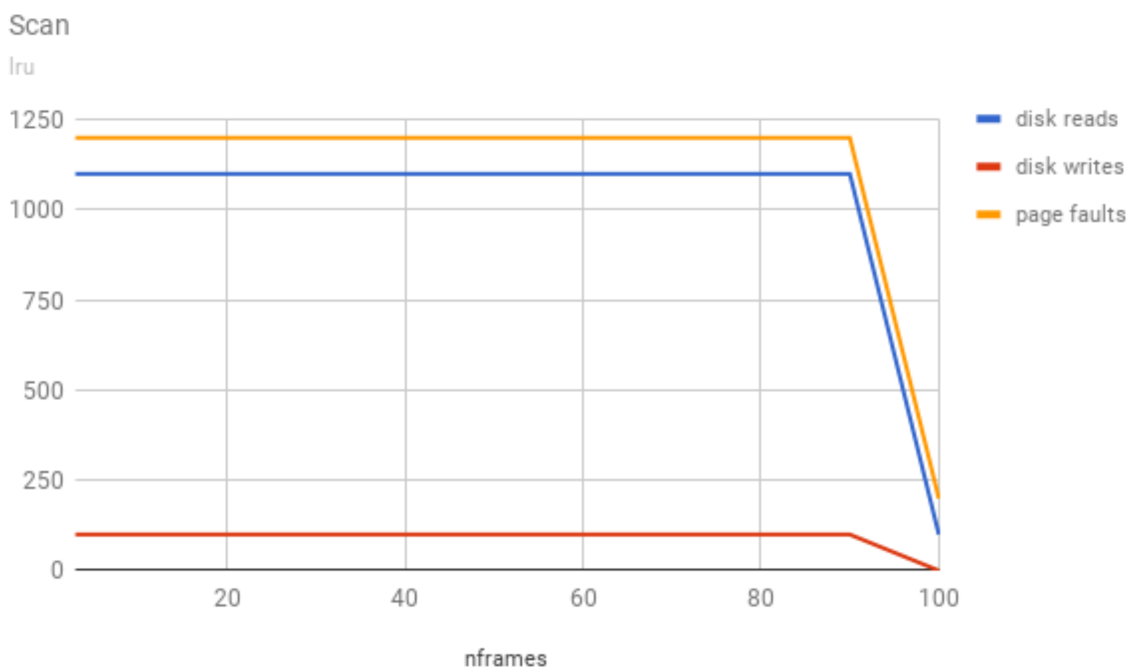

Sort
fifo
3000
2000
1000
0
20
40
60
80
100
nframes
disk reads
disk writes
page faults


Sort
lru
3000
2000
1000
0
20
40
60
80
100
nframes
disk reads
disk writes
page faults

Scan

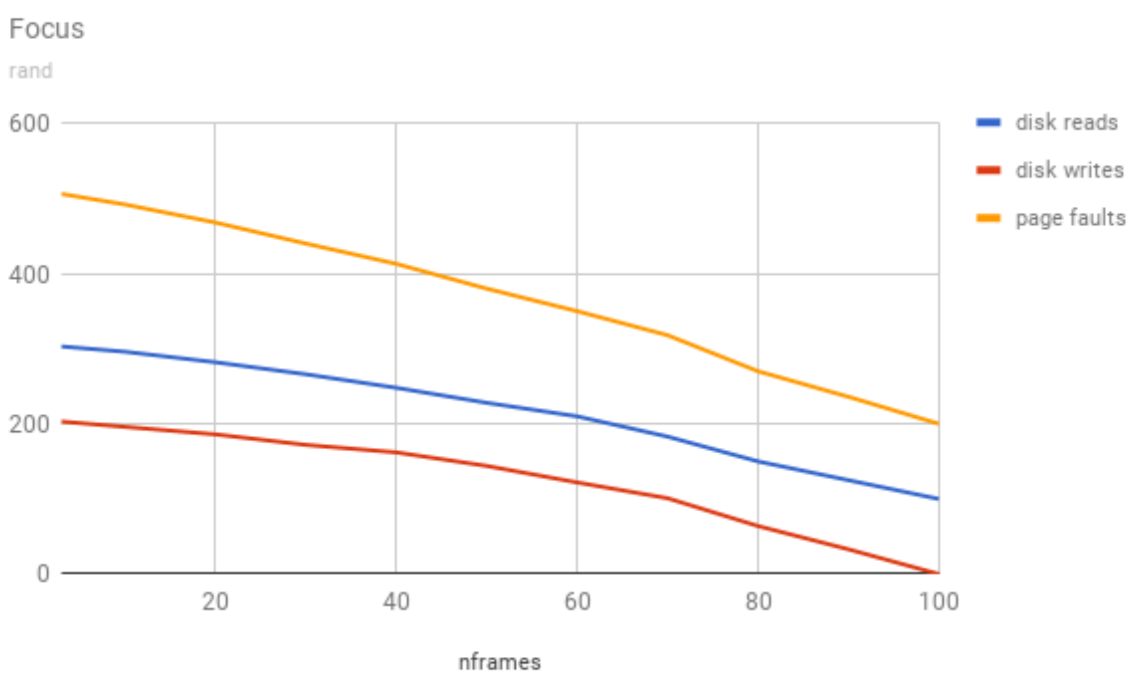

Scan
rand
1250
1000
750
500
250
0
20
40
60
80
100
nframes
disk reads
disk writes
page faults


Scan
fifo
1250
1000
750
500
250
0
20
40
60
80
100
nframes
disk reads
disk writes
page faults

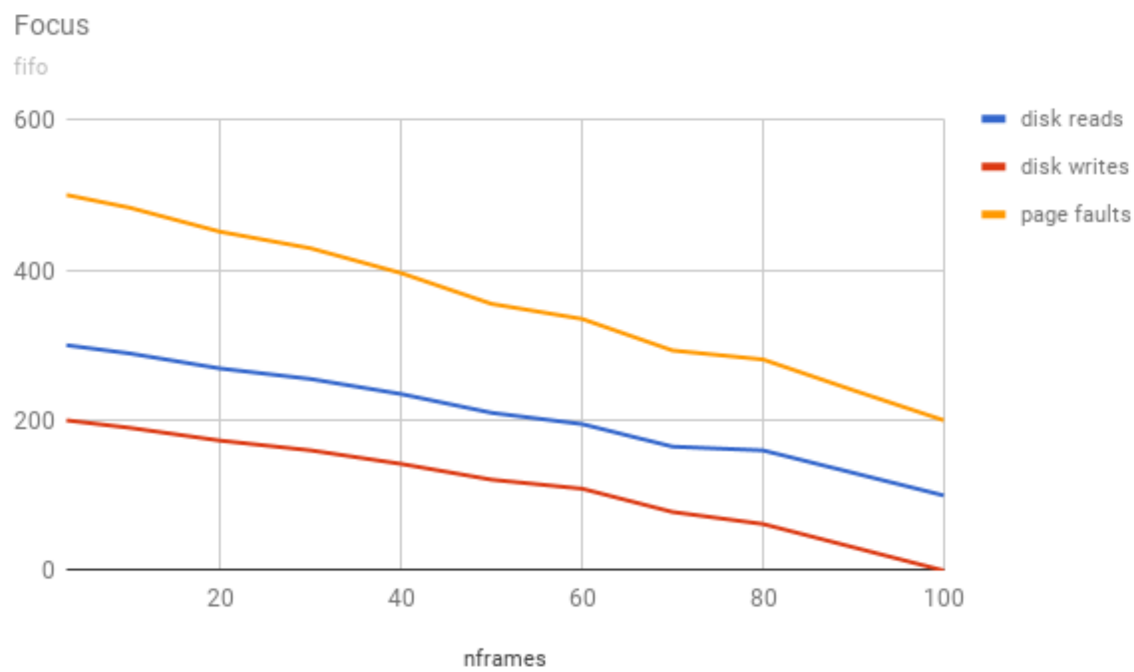


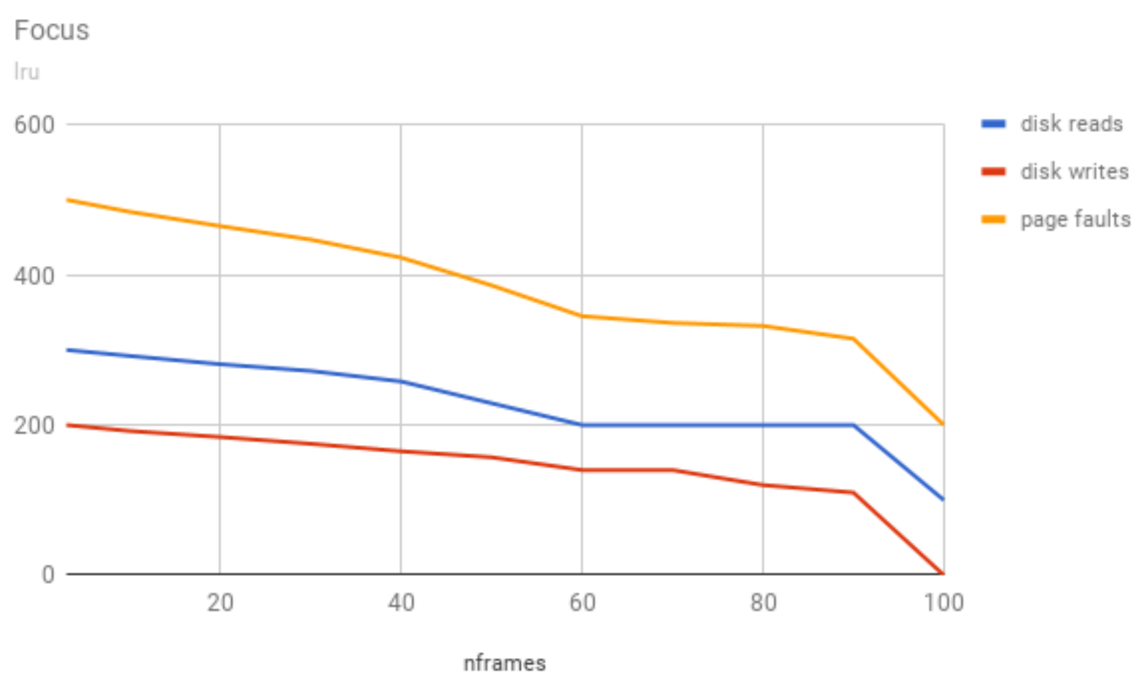
Scan
lru
1250
1000
750
500
250
0
20
40
60
80
100
nframes
disk reads
disk writes
page faults

Focus



Focus
rand
600
400
200
0
20
40
60
80
100
nframes
disk reads
disk writes
page faults





Analysis

Page Fault

If the frame size was equal to the number of pages, then there would be a page fault when first attempting to access data, and then another fault when reading. Therefore, the number of page faults in this case was equal to the number of pages times two.

For the sorting operation, when the algorithm used was rand, the number of page faults increased exponentially from 20 to 3 frames, and decreased relatively linearly from 20 to 100 frames. This was slightly less efficient than the fifo algorithm, since less frames meant that there was a higher chance that the same frame would be replaced multiple times without switching out another page. For sorting, the program uses the quick sort algorithm, which divides elements according to a pivot value and sorts recursively. This algorithm reuses that pivot value in each iteration, but also accesses other elements to compare against that pivot value. Therefore, having about 50 frames would be the maximum efficiency for algorithms that aren't random, as seen in the fifo and lru graphs, where the lines are relatively constant from 50 to 90 frames.

For the scan operation, the number of page faults did not improve for rand from 3 to 20 frames, but was relatively linear when more frames were added. For fifo, the number of page faults was linear from 3 to 60, and then increased a bit more dramatically when more frames past 60 were added. The lru algorithm performed the poorest for the scan operation, with the number of page faults not decreasing at all except when the number of frames equaled the number of pages. The lru program performed poorly since the sum value must have been overwritten repeatedly by the addition of new values in the array used for the scan program.

For the focus function, the number of page faults for rand and fifo followed the same linear decreasing trend. For the lru algorithm, the trend was slighly less dramatic of a slope, though the general trend was the same. There were around 500 page faults when 3 frames were used, decreasing to 200 with 100 frames. The 500 faults can be explained by how the focus program works by using an array proportional to the number of pages, and accessing each data index from that array three times in the code. The first time that data is loaded, there is a page fault, and when data is read another page fault occurs. Then when data is accessed in the future but does no exist in the virtual memory, as will occur if nframes doesn't equal npages, another page fault will be thrown.

Disk Access

If the frame size was equal to the number of pages, then all data was able to fit in the table, and therefore no disk writes had to be made. To load the data into the page table, a disk read had to be made, and so the number of pages requested was equal to the number of disk reads in this particular case.

For the sorting algorithm, there was about double the number of disk reads as disk writes, indicating that when data had to be written to disk, it was reused again later on in the program. This makes sense considering the nature of quick sort, which combines smaller sorted partitions, an operation that requires for elements that had been compared previously to be accessed once more. There was a similar trend as found in page faults, where the number of disk accesses was exponential under 20, and fairly constant over 50, especially for the lru algorithm. The lru algorithm was not too much more efficient than fifo, since elements that had not been accessed in a while would need to be accessed again for the sorting algorithm to work.

Disk writes were much lower for scan than for the focus or sort algorithms. The number of disk reads were proportional to the number of page fauts, and all trend lines closely followed lines plotted for page faults. This indicates that almost every time there was a page fault, there was also a disk read, indicating that the data required for the operation was not stored in the virtual memory. The reason for the low number of disk writes is due to the nature of scan, which creates an array and then scans through that array and adds up a total. Since the data itself is not modified after creation, no additional disk writes are necessary except for incrementing the sum value. The lru program performed poorly since the sum value must have been overwritten repeatedly by the addition of new values in the array. The fifo and rand algorithms improved with more frames as expected, since the array was of length npages*PAGE_SIZE, so the more frames there were, the less often the disk had to be accessed for data, since the info could be stored in virtual memory.

The focus program had about 200 less disk reads as page faults, and about 100 less writes as reads for all three of the algorithms. The focus program first creates an array of size 100, then writes to each index 100 times, before summing the data. Therefore the data is written to 200 times for each index on the disk if removed from virtual memory, and read 100 times more than that due to how the a sum is also taken. This is reflected in the graph where 3 frames are used. When more frames are added, the number of reads and writes decreases relatively linearly. Since data is accessed sequentially, none of the algorithms can provide a perfect solution to minimizing the number of disk accesses.

Custom Strategy

The custom algorithm used was the least recently used algorithm, in which what was untouched for the longest time would be replaced once the page table was full. This algorithm is based on the assumption that data that has been used recently will be used again, so old data was replaced as it was therefore assumed that data that had not been touched in a while would not be as likely to be used again. In order to keep track of the information required for the last recently used algorithm, an array was created to track when a frame was accessed. On every page fault, each index in the array was incremented, while the frame that was accessed in that particular page fault was set to 0. Therefore, once the time came to change out a page in the array, the index with the largest value in the array would be the least recently accessed.